Speech Compression

While the compression techniques presented so far have assumed generic acoustic or visual content, the following chapter presents lossy compression techniques that where especially designed for a particular type of acoustic data: Human speech. Almost every human being on earth talks virtually every day -- needless to say, there is a lot of captured digital speech content. Every movie or TV show contains and audio track, of which usually most of it is spoken language. The most important use of captured speech, however, is for communication, such as in cell-phones, voice-over-IP applications, or as part of video conferencing and meeting recordings. Most of the compression concepts discussed so far will also work on speech. The algorithms presented in the following though will achieve a higher compression ratio wile preserving higher perceptual quality by exploiting speech-specific properties of the audio signal. Therefore, before we dig into the technical details of how speech compression algorithms work, let's first discuss what makes human speech different from generic audio content, such as music or noise.

Properties of Human Speech

The properties of every sound are defined by the properties of the objects that create the sounds, by the environment that the sound waves travel in, and by the characteristics of the receiver and/or capturing device. The object that creates human speech is the vocal tract. Vocal tracts also exist in animals, such as birds or cats. As we all know, the sounds they produce differ substantially from average human speech, so creating a bird-sing compression or cat's meouw encoding algorithm would also be substantially different. Please refer to Chapter XXX for a detailed discussion of human vocal tract physiology. It is important to remind yourself that the human vocal tract can create much more sounds than would generally be classified as speech. Apart from noise imitations (have you never tried to meouw back to a cat?), the most important one is singing. If you ever try to sing into a cellphone, you will notice that the quality isn't really as good compared to regular speech. This is because the methods used in cell phones assume normal, conversational speech, as will be explained in this chapter. Also, we have to say "normal" and "conversational" because human speech can differ significantly in other situations, such as in emotional states of anger, happyness or enragement. Whispering is a yet different form of speech produced by humans that has very distinct properties, as has yelling. The properties of the human voice also changes with health states, such as Alzheimer disease, drug influence, or simply a stuffy nose will change different characteristics of speech.

From a technical perspective, normal, conversational speech differs from general acoustics in the following ways:

•Limited bandwidth: The spectrum and clearness (harmonicity) of the human voice varies with age, gender, and also with the language spoken. While the audible spectrum of sounds is about between 16Hz and 22kHz, speech as generated by humans usually does not go below 80Hz and does not exceed 5kHz. Differences from this general rule might exist when examining individuals. However, it is generally assumed that speech sounds below 80Hz in adult males and below 100Hz in women and children are disregarded by the human ear. The pitch of the human voice is between 120 and 160Hz for adult males and between 220 and 330 Hz for women and children. Vowels, are most important for the intelligibility of speech, they can reach frequencies up to 5kHz. The highest, frequencies are emitted by sibilants, such as “s” or “f”. The frequencies can easily reach into the non-audible spectrum (above 20kHz). Consequently, to capture the whole frequency range of language a 16kHz sampling rate is currently the gold standard, which as of the Nyquest theorem, guarantees the reproducibility of 8kHz. Telephone systems usually use 8kHz, which allows the reproduction of a signal up to 4kHz. This still allows to capture most of the vowels, but consonants and sibilants are only understandable in context. This is why spelling on the phone usually has to be performed in whole words “Alpha”, “Beta”, “Charlie” rather than “a”, “b”, “c”.

•Limited volume: The dynamic of speech is relatively high compared to many other sounds sources, such as some musical instruments, however, in general the volume of human voice is limited to the sound energy the human body can produce. Motorbikes or gun shots can produce a much higher energy level, for example. In a 60 cm distance from the mouth, the typical sound volume of the human voice is about 60dBA. A stronger voice can raise the volume by about 6dB. Yelling measures about 76dBA (males) and 68dBA (females). The sound volume is reduced by about 4dB every time the distance to the microphone is doubled or increased by 4dB when the distance is halved. In general, 16bits per sample are used to represent the dynamics of speech completely. However, so-called plosives like “p” or “t” can easily cause clipping, even with a well-defined capturing environment.

•Limited variance in harmonicity: In contrast to generic audio, speech usually has a certain characteristics governed by the underlying language. This is similar to instruments: A violin for example almost exclusively generated harmonic sounds, while a drum almost exclusively generates inharmonic sounds. Languages usually dictate a certain ratio between vowels and consonants, which translates into a constant ratio of voiced and unvoiced sounds. So, when the language is known, this characteristics can for example be exploited in a predictive coder (see Chapter XXX). In general, the properties of the governing language will also have an impact on the expected dynamics and frequency range of the uttered speech.

The following algorithms all try to exploit the characteristics of speech and have very limited applicability to music or other non-speech. However, all of them are of outmost importance to multimedia computing since they are used by millions of (mostly unaware) people in everyday life.

Target Properties for a Speech Coder

Speech codecs are applied in two main areas: Telephony and voice over IP applications. While the two areas seem to increasingly merge, historically the problems presented themselves in the areas required slightly different solutions. Other applications, such as Internet radio or speech compression for archival purposes yet introduce other priorities.

In general, the targets for a speech coder are:
-Good compression
-Minimum impact on speech intelligibility
-Maximum preservation of speaker characteristics
-Good handling of background noise (e.g. background noise should be eliminated when speech is present, but transmitted when not)
-Low computational complexity (e.g. cell phones needs to be small and should not need)
-Minimum latency, i.e. the delay introduced between transmitter and receiver must be small
-Transmission-error resistance (e.g. packet losses might be concealed)
-Text-independence

Sometimes, language-independence is sacrificed for a more efficient codec when a specific target market is aimed at.

Linear Predictive Coding (LPC)

One of the earliest models used for the compression of speech sounds is the so-called Linear Predictive Coding, or LPC. It is widely used in many places from telephony to military applications (low quality, ultra-low bandwidth) and is part of many, if not most, of todays speech compression algorithms in cell-phones. Typical rates of encoded streams vary from 800bits per second to 16k-bits per second.

As the name implies, LPC is a predictive coder -- and it's linear. The underlying assumption of LPC is that a speech signal is produced by a buzzer at the end of a tube, with occasional added hissing and popping sounds (sibilants and plosive sounds). Although apparently crude, this model is a close approximation to the reality of speech production: The vocal tract forms a tube, which is characterized by its resonances due to its shape and the buzz is produced by the glottis, the space between the vocal folds, and is characterized by its intensity. Some literature calls methods that model signal production for efficient representation parametric redundancy exploitation methods. Figure 1 shows a diagram of one of the earliest LPC algorithms.

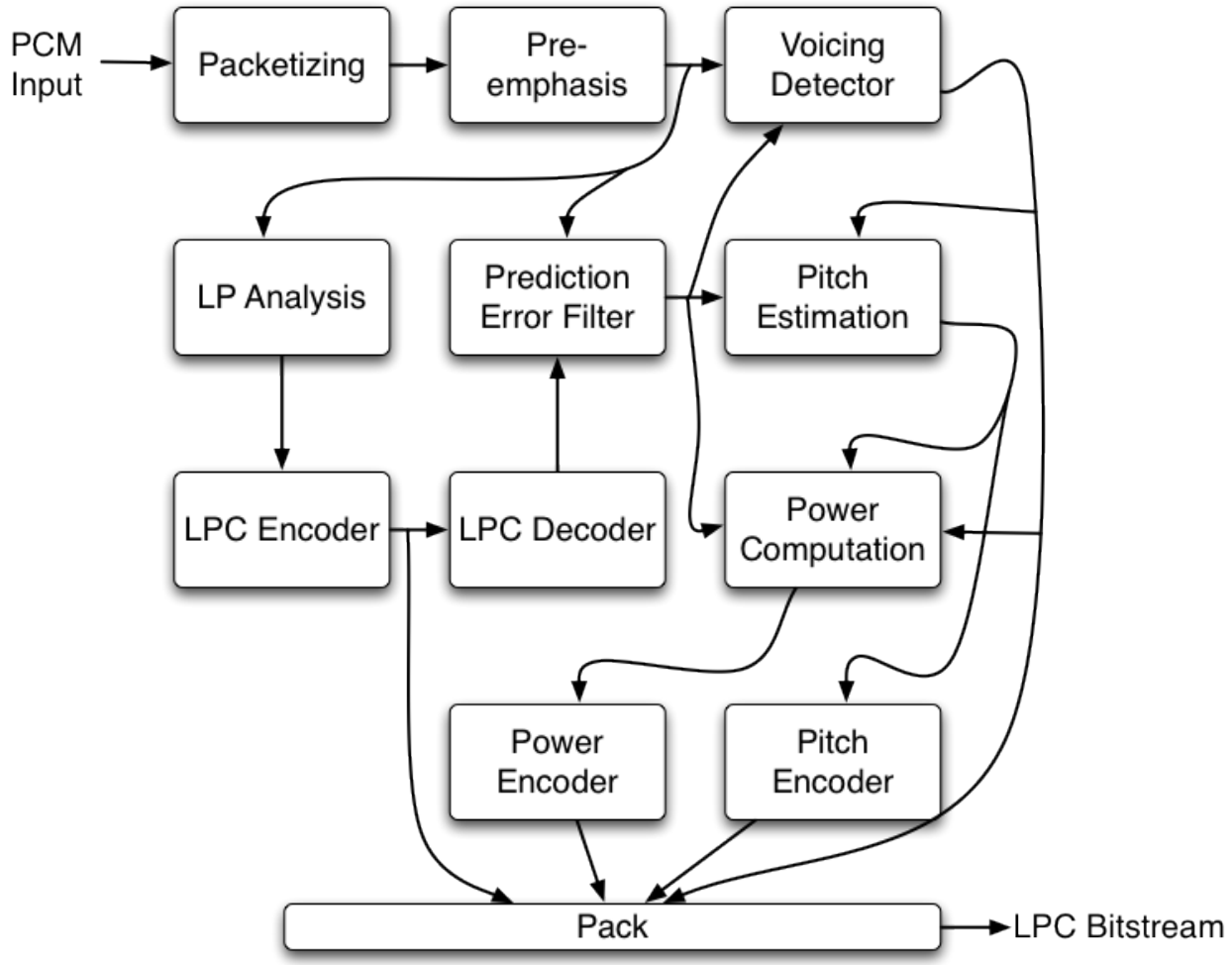

Figure 1. The LPC compression algorithm as defined in the NATO standard FS-1015 (LPC10)

It consists of several steps that are explained as follows. First, the sampled audio signal is packetized into small segments with a typical length of about 10ms. These atomic segments are usually called frames (compare video frames). A pre-emphasis filter increases the magnitude of the higher frequency parts with respect to the lower frequencies in order to increase the quality of the following steps. This trick is used often to increase the perceptual signal-to-noise ratio. This enhanced signal is then used for several steps. First, a voiced/unvoiced detector finds the regions where pitch can be calculated, eg. speech regions that contains mostly vowels. The pitch and the power are then also estimated and later encoded into the bitstream. In parallel, an LP analysis is performed on the signal. The LP signal is then used for error prediction as discussed in the previous chapter. The difference between the LP estimation and the actual signal is encoded together with the power and pitch in the actual bitstream. If the frame is voiced the pitch prediction is estimation from the prediction error signal. The compression achieved usually allows to transmit an 8kHz mono using 2400 bits per second in an understandable quality. The compression works in realtime even on home-sized computers from the 1970s. Not only is the LPC speech compression

a very fundamental algorithm that is used in many different versions in a whole range of speech processing devices, it also allows us to discuss a couple of fundamental speech processing techniques.

Voiced/Unvoiced Detection

How can we determine if a speech frame contains a vowel or not? The methods currently avail-

$$E_{[m]} = \sum_{n=m-N+1}^{m} s^2{}_{[n]}$$

Energy is always a positive number, hence the square root. Alternatively, the absolute value of

$$MSF_{[m]} = \sum_{n=m-N+1}^{m} \left| s_{[n]} \right|$$

As already said, a voiced signal might cross the zero amplitude line more often than a non-

$$SC_{[m]} = \frac{1}{2} \sum_{n=m-N+1}^{m} \left| \operatorname{sgn}(s_{[n]}) - \operatorname{sgn} s_{[n-1]}) \right|$$

The *sgn()* function returns 1 or 0 depending on the sign of the operand. A third method is to cal-

$$PG_{[m]} = 10\log_{10}\left(\frac{\sum_{n=m-N+1}^{m} s^2{}_{[n]}}{\sum_{n=m-N+1}^{m} e^2{}_{[n]}}\right)$$

It can be observed that voiced frames on average achieve 3 dB or more in LPC prediction gain than unvoiced frames, mainly due to the fact that periodicity implies higher correlation among samples, and thus easier to predict. Unvoiced frames, are more random and therefore less predictable. For very low-amplitude frames, prediction gain is normally not calculated to avoid numerical problems; in this case, the frame can be assigned as unvoiced just by verifying the energy level. Thresholding energy, zero-crossing rate and prediction gain is not an exact science. Modern systems use machine learning to find good values for estimating these values on a concrete data set. Finding a perfect boundary between a voiced and an unvoiced segment is nearly impossible.

Pitch Estimation

The fundamental frequency (F0) of a periodic signal is the inverse of its period. The subjective "pitch" of a sound usually depends on its fundamental frequency but also depends on other factors. Given that a frame is voiced, estimating the pitch of the frame, however, is one of the most important and frequently demanded operations in audio processing. In speech processing, the pitch period is defined as the time between successive vocal cord openings. It is important to note that expected values for the for men, possible pitch periods lie between 4ms and 20ms (frequency between 50Hz and 250Hz) and for women and children between about 2ms to 8ms (frequency between 120Hz and 500Hz). Unfortunately, estimating this time is, like the voiced/unvoiced detection, not an exact science either. Unless the signal is artificially generated, pitch period estimation is a complex undertaking due to the lack of perfect periodicity in real world signals. Unless trained thoroughly, even a singer's voice has no perfect pitch due to interference with formants of the vocal tract (voice impurities). Also, the uncertainty of the starting point of a voiced segment (see previous paragraph) and other real world problems, such as noise and echo make perfect pitch estimation hard. For this reason, arbitrarily complex pitch estimation algorithms have been developed and refinements will still be presented in research papers to come. In practice, pitch period estimation is implemented as a trade-off between computational complexity and performance. From the many algorithms that have been proposed for this task, only two will be presented here. The first, and most frequent method, is the so-called autocorrelation method.

$$R_{[l,\,m]} = \sum_{n=m-N+1}^{m} S[n]S[n-l]$$

The autocorrelation value reflects the similarity between the frame *s[n]* and the time-shifted version *s[n-l]*. Again, the frame has length *N* containing and is ending at instant *m.* The variable *l* is a positive integer representing a time lag and *n= [m–N+1,m].* The range of lag is selected so that it covers a wide range of pitch period values. For example, at 8kHz sampling rate, if *l* is between 20 and 147 (2.5 ms to 18.3 ms), the possible pitch estimation times values range from 54.4 Hz to 400 Hz. By calculating the autocorrelation values for the entire range of lag, it is possible to find the value of *l* associated with the highest autocorrelation representing the pitch period estimate. In other words, the autocorrelation *R* is maximized when the lag *l* is equal to the pitch period.

The pseudo-code for the algorithm would look like this:

```
// Input: last index in frame m, number of samples N,
// sampling rate sr per second
// Output: The pitch period in seconds.
pitch(m, N, sr)
    peak := 0
    FOR l:=20 TO 150
        autoc:=0
        FOR n:=m-N+1 TO m
         autoc:=autoc+s[n]*s[n-l]
    IF (autoc>peak)
        peak:=autoc
        lag:=l
pitch := lag/sr
```

A second method for pitch estimation is the so-called Magnitude-Difference Function. The idea is that for short segments of voiced speech it is reasonable to expect that s[n]-s[n-l] is small for l =0,±T,±2T, . . , with T being the signal's period. By computing the MDF for the lag l range of interest, we can estimate the period by locating the lag value associated with the minimum magni-

$$MDF[l,\,m] = \sum_{n=m-N+1}^{m} \left| S[n] - S[n-l] \right|$$

tude difference. And here is the equation for the MDF:
Note that from the same equation, each additional accumulation of term causes the result to be greater than or equal to the previous sum since each term is positive. Thus, it is not necessary to calculate the sum entirely: if the accumulated result at any instance during the iteration loop is greater than the minimum found so far, calculation stops and resumes with the next lag. Also, no multiplication is involved in this method which is sometimes interesting for speed and memory

usage on small devices. Overall, this method is faster than the regular autocorrelation method. The idea is implemented with the following pseudocode:

```
// Input: last index in frame m, number of samples N, sampling rate sr per
second
// Output: The pitch period in seconds.
pitch2(m, N, sr)
    min := infinity
    FOR l:=20 TO 150
        mdf:=0
        FOR n:=m-N+1 TO m
            mdf:=mdf+ABS(s[n]-s[n-l])
            IF (mdf>=min) BREAK
        IF (mdf<min)
            min:=mdf
            lag:=l
pitch2 := lag/sr
```

It is important to note here, that the two methods present really only estimate pitch. In fact, nobody will probably ever be able to accurately calculate pitch, since pitch, even though depending on the fundamental frequency, is subjective. Even when focussing only on speech, there are many factors that distort pitch. For example, periodic vibration at the glottis may produce speech that is less perfectly periodic because of movements of the vocal tract that filters the glottal source waveform. Then, glottal vibration itself may also show aperiodicities, such as changes in amplitude, rate, or glottal waveform shape. Reverberation inside the vocal tract also distort pitch.

LP Analysis

The underlying model for LPC is this:

$$\tilde{x}(n) = a_1 x(n-1) + a_2 x(n-2) + \cdots + a_M x(n-M) = \sum_{i=1}^{M} a_i x(n-i)$$

where x˜(n) is the prediction of the present sample generated through linear combination of the past M samples x(n) to x(n-i). The a_i are called the linear prediction coefficients. In other words the predictor tries to predict a sample as a linear combination of the previous outputs. Usually 10-32 linear prediction coefficients are used. The number of coefficients is usually chosen depending on the frequency used, for 8kHz sampling rate 10-dimensional LPC analysis is usually good, for 16kHz, 20 is a typical value.

The idea is then to minimize the error between the actual and the predicted value:

$$\varepsilon(n) = x(n) - \tilde{x}(n) = x(n) - \sum_{i=1}^{M} a_i x(n-i)\,.$$

This is usually solved by mathematical optimization, such as Levinson-Durbin recursive method, the scientific literature also discusses other methods (see research references). For an order *N* filter, the filter coefficients a_i are found by solving the $N \times N$ linear system $Ra=r$, where

$$\mathbf{R} = \begin{bmatrix} R(0) & R(1) & \cdots & R(N-1) \\ R(1) & R(0) & \cdots & R(N-2) \\ \vdots & \vdots & \ddots & \vdots \\ R(N-1) & R(N-2) & \cdots & R(0) \end{bmatrix}$$

$$\mathbf{r} = \begin{bmatrix} R(1) \\ R(2) \\ \vdots \\ R(N) \end{bmatrix}$$

with *R(m)* being the auto-correlation of the signal *x[n]*, computed as described above (and implemented below):

$$R(m) = \sum_{i=0}^{N-1} x[i]x[i-m]$$

and *r* being the so-called reflection coefficient. The following pseudo-code illustrates a practical implementation of this method which was first invented by N. Levinson in 1947 and then modified by J. Durbin in 1959 (see research references).

```
// Compute LPC coefficients from a series of auto-correlation coefficients
// Input:  dim order of LPC analysis,
//         ac [0...dim+1]autocorrelation values (see helper function below)
// Output: ref[0...dim] reflection coefficients R(N),
//         lpc[0...dim] LPC coefficients,
//         error        residual error
levinson_durbin(dim, ac)
      error := ac[0]
      IF (error == 0)
            set all ref[i]:=0
            levinson_durbin := ([0...0],[0...0],0)
            // return and exit routine
      // main loop
      FOR i:=0 TO dim-1
            // Calculate the reflection coefficient
            r:=-ac[i+1]
            FOR j:=0 TO i
                  r:=r-lpc[j]*ac[i-j]
                  r:=r/error
                  ref[i]:=r
            // Update LPC coefficients and total error
            lpc[i]:=r
            FOR j:= 0 TO (i/2)-1
                  temp := lpc[j]
                  lpc[j] := lpc[j]+r*lpc[i-1-j]
                  lpc[i-1-j] := lpc[i-1-j]+r*temp
            IF (i%2 == 1)
                  lpc[j] := lpc[j]+lpc[j]*r
```

```
            error := 1.0-r*r
levinson_durbin := (ref,lpc,error)
```

The autocorrelation values can be calculated as follows:

```
// Compute the autocorrelation coefficients needed for the LPC
// Input:  n number of audio samples,
//         x[0..n-1] audio samples
//         lag a maximum lag range
// Output: ac[0...lag-1] autocorrelation values
lpc_autocorrelation(n, x, lag)
      WHILE (lag>0)
            lag:=lag-1
            FOR i:=lag TO n
                  d:=0
                  d:=d+x[i]*x[i-lag]
            ac[lag]:=d
lpc_autocorrelation:=ac
```

With the increasing availability of multiple CPU core architectures, an alternative solution to the Levinson-Durbin recursion is in frequent use that is easier to parallelize. The so-called Schuer recursion is related to the Levinson-Durbin method but faster on parallel architectures. On multiple cores, Levinson-Durbin would take time proportional to *O(dim*log(dim))*, Schuer requires time proportional to *dim.* The following pseudo code, which again relies on *lpc_autocorrelation,* shows the idea:

```
// Alternative recursion algorithm for parallel architectures
// Input:  dim order of LPC analysis,
//         ac [0...dim+1]autocorrelation values (see helper function below)
// Output: ref[0...dim] reflection coefficients R(N),
//         error        residual error

schuer(dim,ac)
      error := ac[0]
      IF (error == 0)
            set all ref[i]:=0
            schur := ([0...0],0)

      // Create a so-called generator matrix G with dimensions [2,dim]
      FOR i := 0 TO DIM-1
            //  Calculate this iteration's reflection coefficient and error.
            G[0][i] := ac[i+1]
            G[1][i] := ac[i+1]
      i:=0
      WHILE (true)
            r := -G[1][0]/error
            ref[i] := r
            error := error + G[1][0]*r
            i:=i+1
            IF (i>=dim)
                  schur := (ref,error)
```

```
                    // return and exit routine
               // Update the generator matrix.
               // Unlike the Levinson-Durbin summing of reflection coefficients,
               // this loop could be distributed to many processors which
               // each take only constant time.
               FOR m := 0 TO dim-i-1
                    G[1][m] := G[1][m+1] + r * G[0][m]
                    G[0][m] := G[1][m+1] * r + G[0][m]
schur := (ref,error)
```

The calculation of the related LPC coefficients is left as an exercise.

As mentioned above, the building blocks introduced here for explaining the LPC algorithm have been reused many times in other speech compression algorithms. LPC modeling is also used for speech synthesis. For example, the very popular "speak'n'spell" toy from the 1980s (see web references) has used LPC for reading words. LPC can also be used as a feature in speech analysis, eg. for speech or speaker recognition. Most importantly though, the LPC algorithm was used as a basis for further, more complex algorithms for speech compression, of which some are described in the next sections.

CELP

CELP is an enhancement of the LPC compression providing better quality speech than LPC-10e, described above, without increasing the bit rate too much. Since LPC is a synthesizer, a natural extension is to use a wavetable (see Chapter XXX) to increase the quality, trading off the increase of coder and decoder complexity for smaller bitrates. Therefore, many successful methods use a so-called codebook, usually a table of typical residue signals. In a nutshell, the analyzer compares the residue to all the entries in the codebook, chooses the entry which is the closest match, and just sends a reference to that entry. The synthesizer receives the reference, retrieves the corresponding residue from the codebook, and uses the "code" to "excite" the re-synthezised signal, hence the name Code Excited Linear Prediction (CELP). The principle behind CELP is also called analysis by synthesis because the encoding (analysis) is performed by perceptually optimizing the decoded (synthesis) signal in a closed loop.

CELP search is broken down into several steps. Typically, encoding is performed in the following order:

1.Linear prediction coefficients are computed, converted to Line Spectral Frequencies (see below), and then quantized
2.An adaptive codebook is searched and its contribution removed
3. A fixed codebook is searched

The steps will be explained in the following.

LSF encoding

An essential trick is to transform the linear prediction coefficients into so-called line spectral frequencies (LSF), sometimes also called line spectral pairs (LSP).

The linear prediction polynomial (from above) can also be written as

$$A(z) = 1 - \sum_{k=1}^{p} a_k z^{-k}$$

which can be decomposed into the following two complex equations:

$$P(z) = A(z) + z^{-(p+1)}A(z^{-1})$$
$$Q(z) = A(z) - z^{-(p+1)}A(z^{-1})$$

where *P(z)* is said to correspond to the vocal tract with the glottis closed and *Q(z)* with the glottis open. The reason for this transformation is to exploit the following mathematical trick.

A(z) has complex roots anywhere within the unit circle (z-transform) but *P(z)* and *Q(z)* have the very useful property of only having roots directly on the unit circle. So in order to find the roots, one takes a test point $z = exp(jw)$ and evaluates $P(exp(jw))$ and $Q(exp(jw))$ using a grid of points between 0 and π. The zeros of P(z) and Q(z) also happen to be interspersed which is why one swaps coefficients as one find roots. Therefore the process of finding the LSP frequencies is finding the roots of two polynomials of order *p+1*.

And here is why this helps: LPC coefficients do not quantize well since small quantization errors may lead to large spectral distortion. Of course, the higher order bits in the representation of the coefficients are naturally more sensitive to transmission errors than the lower-order ones. Also the LPC coefficients do not interpolate well, i.e. one cannot compute them at two distinct times and expect to accurately predict values in between. Therefore, instead of encoding the coefficients, it would be better to encode the zeros of the LPC equation. However, finding these zeros numerically entails a computationally complex two-dimensional search, while the zeros of *P(x)* and *Q(x)* can be found by simple one-dimensional search techniques. Over the years, it has been found that LSP frequencies quantize well and interpolate better than all other parameters that have been tried in speech applications.

To convert back to LPCs, one evaluates *A(z) = 0.5[P(z) + Q(z)]* by putting signal samples through it the order of the LPC times, yielding back the original *A(z)*.

Adaptive Codebook

The entries in the adaptive codebook consist of delayed versions of the excitation to make it possible to efficiently code the portions of the signal that are periodic, such as the voiced parts of speech. In the decoder, the final excitation is produced by summing the contributions from the adaptive codebook and the fixed codebook (see below):

$$e[n] = e_a[n] + e_f[n]$$

where $e_a[n]$ is the adaptive codebook contribution and $e_f[n]$ is the fixed codebook contribution. The filter that shapes the excitation has an all-pole model of the form $1/A(z)$, where $A(z)$ are obtained using linear prediction. An all-pole filter is used because it is a good representation of the human vocal tract and because it is easy to compute.

Fixed Codebook

A good choice for a fixed codebook is normal distributed random vectors. The reason for this is that the LPC-filtering and the adaptive codebook already removes large parts of the inter-dependencies between the samples yielding relatively white-noise-like residual. Usually, the codebook comprises of about 1024 excitation vectors (10-bit codebook) for a 40-sample subframe with 8 kHz sampling frequency.

The methods for finding the right entry in the adaptive codebook vary from coder to coder. Many coders simply uses the Euclidian distance to determine the similarity between code vectors. Speex uses the Euclidian distance shaped with a perceptual weighting functions that tries to roughly approximate noise perception in the human ear. Figure 2 shows the optimization loop.

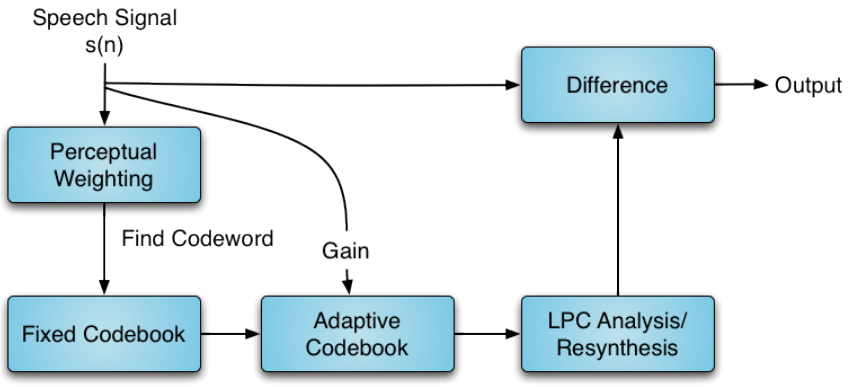

Figure 2. The CELP encoder loop performing analysis by synthesis until the perceptually weighted difference is minimal.

In the end, the encoder transmits 144 bits of information based on 240 audio samples of speech (30 ms). The bit allocation is shown in Table 1.

Parameters	No. of bits
LSF	34
Adaptive Filter	48
Fixed Codebook Index	36
Gains	20
Synchronization	1
Error Correction	4
Future Use	4
Total	**144**

Table 1. A typical CELP bitstream.

The CELP algorithm is most prominently described in the NATO standard FS 1016 which provides good quality, natural sounding speech at 4800 bit/s and in ITU-T recommendation G.728 operating at 16 kbit/s. The popular open source speech codec Speex (see web references) is also based on CELP. Speex is targeted at voice over IP applications.

GSM

GSM stands for Global System for Mobile communications and was originally developed by a group called Groupe Spécial Mobile with the same acronym. The group was formed by the Conference of European Posts and Telegraphs (CEPT) in 1982 in an effort to develop a pan-European public land mobile system. Today, GSM is the number one standard for mobile phones in the world. The GSM Association, estimates that 80% of the global mobile market uses the standard. Its ubiquity, which now spans the world, makes international roaming easily possible and enables subscribers to use their phones in many parts of the world. The main difference between GSM and its predecessors in that both signaling and speech channels are digital, and thus is considered a second generation (2G) mobile phone system. This also makes data transmission over the same line very easy, enabling services such as text and multimedia messaging. The GSM standard describes more than just voice compression. It specifies everything necessary to build a

global communication infrastructure, such as the structure of the radio network, the frequency ranges, the antennas and cells, subscriber identification mechanisms, security standards, power control, and so on. For further information refer to the references and Chapter XXX. This section only provides a brief overview of the speech codecs defined in GSM, which are mainly based on LPC.

GSM has used a variety of voice codecs to compress 3.1 kHz sampling rate audio captured by the cell phone into between 6.5 and 13 kbit/s. In the original specification, only a so-called Half Rate (5.6 kbit/s) and Full Rate (13 kbit/s) codec were defines. These used a system based on linear predictive coding.

In 1997, the Enhanced Full Rate (EFR) codec working on a 12.2 kbit/s basis was introduced. With the development of UMTS, this codec was modified into the so-called AMR (Adaptive Multi-Rate) codecs. This is a variable-rate codec which is high quality and robust against interference when used on Full Rate channels, and less robust but still relatively high quality when used in good radio conditions on Half Rate channels.

Even though still built from almost exclusively the concepts explained in this chapter, current codecs, such as GSM-AMR have grown in complexity beyond a size that can be presented in pseudo-code in this book. We therefore limit ourselves to presenting and discussing the block diagram, shown in Figure 3.

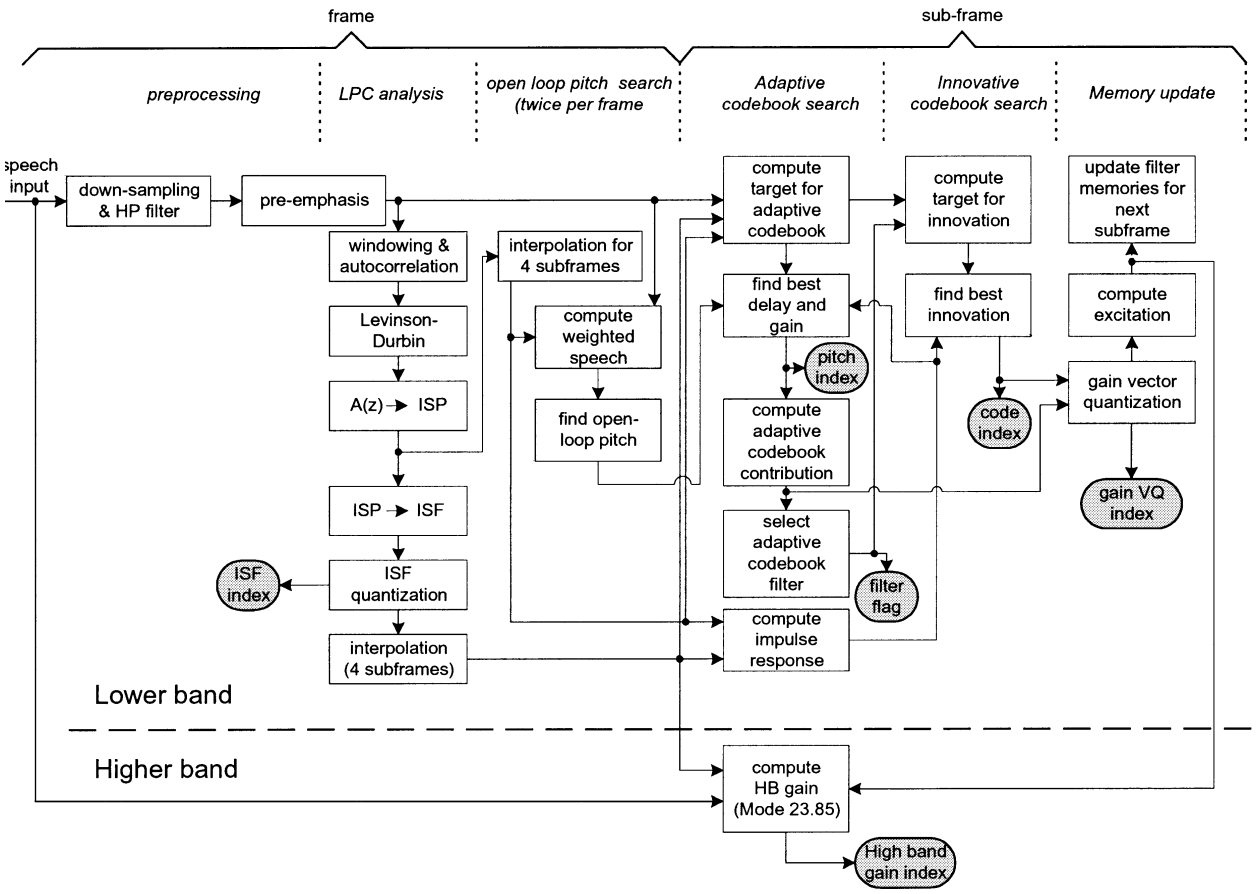

Figure 3. The structure of the GSM-AMR used in millions of cell-phones today (diagram by Bessette et al.).

The GSM AMR-WB codec is based on a special version of the CELP codec, the so-called Algebraic Code Excited Linear Prediction (ACELP) algorithm. ACELP is uses an algebraic adaptive codebook, i.e. the codebook entries are not stored explicitly but as mathematical formulas. The main advantage is that the codebook it uses can be made very large (> 50 bits) without running into storage (RAM/ROM) or complexity (CPU time) problems. The main disadvantage is that the technology is patented and is therefore not freely available. Although the ACELP coder gives very good performance on narrow-band signals, some difficulties arise when applying the telephone-band optimized ACELP model to wideband speech, therefore additional features needed to be added to the model for obtaining high quality on wideband signals. The GSM-AMR codec works in different modes for different bandwidth. The bitrates are 23.85, 23.05, 19.85, 18.25, 15.85, 14.25, 12.65, 8.85 and 6.6 kb/s.

The input signal is down-sampled and pre-processed using a high-pass filter and a noise-reduction filter. The ACELP algorithm is then applied to the down-sampled and pre-processed signal. Linear Prediction analysis is then performed once per 20 ms frame. The set of linear prediction parameters is converted to, so-called immittance spectrum pairs (ISP) (see research references),

which is a different form of LSF, and quantized using vector quantization (VQ). The speech frame is divided into subframes. The adaptive and fixed codebook parameters are transmitted every subframe. The pitch lag is encoded with 9 bits in odd subframes and relatively encoded with 6 bits in even subframes. Another bit per subframe is used to determine the low pass filter applied to the past excitation. The pitch and algebraic codebook gains are jointly quantized using 7 bits per subframe.

The highest frequency band (6400–7000 Hz) is reconstructed in the decoder using the parameters of the lower band and a random excitation. No information about the higher band is transmitted, except in the best mode (23.85kb/s), where the higher band gain is transmitted using 4 bits per subframe. In other modes, the gain of the higher band is adjusted relative to the lower band using voicing information. The spectrum of the higher band is reconstructed by using a wideband LP filter generated from the lower band LP filter.

As a result, the decoder has a similar structure, as shown in Figure 4.

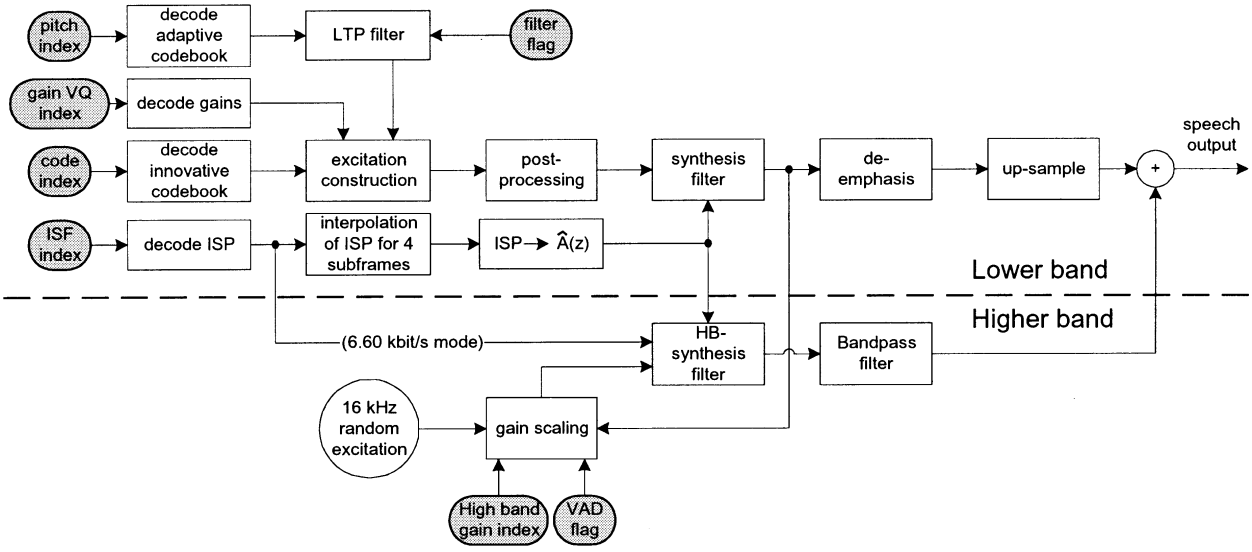

Figure 4. The structure of the GSM-AMR used in millions of cell-phones today (diagram by Bessette et al).

Not only mobile phones use LPC variants, Skype's SVOPC and the later SILK codec use them too. SVOPC and Silk are also tuned to conceal frame drops. This is done by interpolating LSF parameters between two received frames to make up for the missing one.

All speech codecs have in common that they use a model of speech production to save bits. The next chapter will explain perceptual codecs. These use a model of perception. Of course, these

codecs can be used for speech compression as well, although, as of today, speech-production coders still seem to do better on speech than perceptual coders.

Exercises

1.Elaborate which target properties for a speech coder would be most important for: A regular phone, a cell phone, Internet radio, and an Internet teleconferencing application.
2.Describe how speech intelligibility differs from audio quality.
3.Implement a voiced/unvoiced detector. Then try it on different voice recordings and describe the limits of your approach
4.Implement a pitch estimator. Then try it on different voice and music recordings and describe the limits of the approach.
5.Perform a runtime-analysis of the Levins-Durbin and Schuer algorithms on a single CPU. Then describe how the Schuer algorithm performs better on multiple CPUs. Calculate the runtime for it on different CPUs.
6.If one where to use LPC to model music, which types of instruments would be modeled well and which ones would not? Give examples and explain why.
7.Write the (pseudo-)code for calculating the LPC coefficients from the reflection coefficients as output from the Schuer pseudo-code presented in this chapter.
8.Write (pseudo-)code for calculating the LSF coefficients from LPC coefficients.
9.Write (pseudo-)code to implement a CELP encoder.
10.Use a current implementation of a speech encoder (e.g. in your cell phone or using a voice-over-IP application) and transmit speech, music, and noise through it. Describe and explain the artifacts observed.
11. Given the models presented in this chapter, discuss what other elements are contained in speech that were not discussed. Describe how these could be handled.
12.Use a speech compression algorithm of your choice and describe how you would handle packet loss?

Literature

P. Vary, R. Martin: Digital Speech Transmission: Enhancement, Coding and Error Concealment, Wiley, 2006

Web Links

Speak'n'spell history page: http://www.speaknspell.co.uk/
Speex: http://www.speex.org/
GMS World: http://www.gsmworld.com

Comp.Speech FAQ: http://www.speech.cs.cmu.edu/comp.speech/

Research Papers

-Levinson, N. (1947). "The Wiener RMS error criterion in filter design and prediction." *J. Math. Phys.*, v. 25, pp. 261-278.

-Durbin, J. (1960). "The fitting of time series models." *Rev. Inst. Int. Stat.*, v. 28, pp. 233-243.

-Trench, W. F. (1964). "An algorithm for the inversion of finite Toeplitz matrices." *J. Soc. Indust. Appl. Math.*, v. 12, pp. 515-522.

-Bessette, B. and Salami, R. and Lefebvre, R. and Jelinek, M. and Rotola-Pukkila, J. and Vainio, J. and Mikkola, H. and Jarvinen, K. : “The adaptive multirate wideband speech codec (AMR-WB)”, IEEE Transactions on Speech and Audio Processing, Vol. 10, No. 8, pp. 620-636, 2002.

-Bistritz, Y. Peller, S.: “Immittance spectral pairs (ISP) for speech encoding”, Proceedings of IEEE ICASSP, page(s): 9-12, April, 1993.

-B.S. Atal, "The History of Linear Prediction," *IEEE Signal Processing Magazine*, vol. 23, no. 2, March 2006, pp. 154–161.

-M. R. Schroeder and B. S. Atal, "Code-excited linear prediction (CELP): high-quality speech at very low bit rates," in *Proceedings of the IEEE ICASSP*, vol. 10, pp. 937–940, 1985.

-Xianglin, Wang; C.-C. Jay Kuo (May 1998). "An 800 bps VQ-based LPC voice coder". *The Journal of the Acoustical Society of America* **103** (5): 2778.